MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2019/2020 SESSION

ECE3206 – OBJECT ORIENTED PROGRAMMING WITH C++

(CE, EE, ME & TE)

12 MARCH 2020
9:00 AM – 11:00 AM
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of six (6) pages only (including this page).
2. **This is an open-book exam**, so the questions are more in-depth and there are fewer questions than normal exams. Each student is allowed to bring in one book and one stapled notes into the exam room.
3. There are **THREE (3) QUESTIONS** in this paper. Answer **ALL QUESTIONS**.
4. All programming questions must be answered using the C++ language.
5. Write your answers in the Answer Booklet provided.
6. State all assumptions clearly.

Question 1

The following code snippet in Figure Q1.1 shows the definition of a class that represent a typical bank transaction. The related member functions allows new account to be created, money deposited and withdrawn from the account respectively. In addition, money transfer between accounts is also supported.

```
class  BankTransaction {

private:
      float balance;
      string name;

public:
      BankTransaction() { balance = 0; }
      BankTransaction(string a_name, float a_amount);
      float getBalance() { return balance; };
      string getName() { return name; }
      bool withdrawMoney(float amount);
      bool transferMoneyTo(BankTransaction& acc_dest,
float amt);
      void chargeAnnualFee(float annual_charge);
      bool depositMoney(float amount);
};
```

Figure Q1.1

(a) Implement all the member functions and constructor for class **`BankTransaction`**. All functions must be defined outside of the class definition.

i. `BankTransaction(string a_name, float amount)` constructor initialize member variables **name** based on `a_name` and **balance** based on a_amount to be deposited into the account.

[3 marks]

ii. The `getBalance()` and `getName()` functions are the getter member functions that retrieve the **balance** in the account and customer **name**.

[4 marks]

iii. The `depositMoney(float amount)` member function add **`amount`** of money into the account and update member variable **`balance`**.

[4 Marks]

Continued ...

iv. The `function transferMoneyTo(BankTransaction& acc_dest , float amt)` transfers **amt** into **`acc_dest`** account from source account. The source account is from object that invoke the function. The function return true if sufficient balance is available for transfer. If not, it return false. The balance for transferor and recipient should be updated accordingly.

[4 Marks]

v. The `chargeAnnualFee(float annual_charge)` member function applies the annual fee of **annual_charge** percentage of the balance in account. The balance is updated after deducting annual fee charge.

[4 Marks]

vi. The `withdrawMoney(float amount)` function allows customer to withdraw **`amount`** of money, update the **`balance`** and return true if the amount to be withdrawn is less than **`balance`**.

[6 marks]

(b) i. Draw the Unified Modeling Language (UML) class diagram to describe the class BankTransaction.

[4 mark]

ii. Implement the `main()` function to test the member functions that you have implemented. It should display the following output. Assume 20% rate of the balance as the annual fee.

```
New account
Name =Ejen Ali  Account balance = 3000

Deposit RM 200  -> Account balance = 3200

Withdraw RM 300          -> Account balance = 2900

New account
Name =Jason     Account balance = 1500

Transfer RM600 from Ejen Ali to Jason
Name =Ejen Ali  Account balance = 2300
Name =Jason     Account balance = 2100

Name =Jason
Account balance after deducting annual fee for Jason = 1680
```

[16 Marks]

iii. Describe a method with C++ code to write the output listed in b(ii) to a text file.

[5 marks]

Continued...

Question 2

(a) Use the following class definition in Figure Q2.1 as your reference. The program shows the relationship between multiple classes for representing different shapes in computer graphic.

```
#include <iostream>
using namespace std;
enum colourOption {black,blue,red};
class Shape {
  protected:
  colourOption colour;
  int lineThickness;
  public:
  Shape(colourOption c, int line) {colour=c; lineThickness=line;}
  virtual void calculateArea() = 0;
};

class Rectangle : public Shape {
  int width, height;
  public:
  Rectangle(colourOption c, int line_t, int a_width, int a_height);
  virtual void calculateArea();
};

class Square : public Shape {
  int length;
  public:
  Square(colourOption c, int line_t, int len) ;
  virtual void calculateArea();
};

// Use the following driver function as your reference
int main() {
  colourOption c=black;
  Square sq(c, 2, 80);
  sq.calculateArea();
  Rectangle rec(c,2,9,8);
  rec.calculateArea();
}
```

Figure Q2.1

Continued...

(a) i) Draw the UML class diagram based on the class definition in Figure Q2.1. The member variables and methods are not required in the class diagram.

[4 Marks]

ii) Implement the parameterized constructor for the **Square** class and **Rectangle** class.

[4 Marks]

iii) Explain the definition of polymorphism in object oriented programming and the benefit of using the polymorphism in C++ program.

[3 Marks]

iv) Implement the member function to `calculateArea()` for both **Square** and **Rectangle** classes and display the result on the screen. Return the **area** value

[4 Marks]

v) In the `main()` function, write a short C++ code to create an array that store 3 **Shape** objects based on Table Q2. Then, total up the area for all the 3 Shape objects. Use an array of pointers **ptr** to **Shape** object to represent the objects. The pointer is declared as Shape* ptr[3]

[5 Marks]

Table Q2

Type of shape	width	height
Square	50	
Rectangle	120	40
Square	45	

b)

A complex number z is represented by the real part (a value) and imaginary part (b value). The complex number is written in the format z= a + bj.

Declare a class **Complex** to store the complex number and provide implementation for operator functions that overload the >, + and – operators so that it can be used on **Complex** objects. The > operator return the string "true" if the complex number magnitude of the left operand is higher than the right operand, else it return the string "false". Implement the member function **`output()`** that display the complex number in a+bj format. Assume c, c1 and c2 are three **Complex** objects with c1 = 3+4j and c2 = 5+7j . Use the following test program as your reference.

```
int main()
{
        Complex c1(3, 4), c2(5, 7), c;
        c = c1 + c2;
        c.output();
        c = c1 - c2;
        c.output();
        cout << "\n\n c1>c2 --> " << (c1>c2);
        return 0;
}
```

[10 Marks]

Continued...

Question 3

The following test program produces the output in Figure Q3.1. In order to support the test program and the required output, declare the template class **Container** and implement all the member functions for the class. Use the test program and its output as your reference. The program uses template class **Container** to create object **c1** that store an array of **int** numbers. The **Container** class uses member variable **T* ptr** data type to store an array of numbers. The member function **setData(T val)** is a member template function to insert the data into the array managed by **ptr.** The member function **displayData()** display the output as shown in figure Q3.1. The generic data type is represented by **T**.

```
int main() {
      Container<int> c1(5); // container size=5
      c1.setData(20);
      c1.setData(30);
      c1.displayData();
      cout << "\n\t Number of data in container  ="
           << c1.getNumdata();
      cout << endl;
      c1.setData(66);
      c1.displayData();
      cout << "\n\t Number of data in container  ="
           << c1.getNumdata();
      cout << endl << endl;

      getchar();
      return 0;
}
```

```
The data in container ->20  30
Number of data in container  =2

The data in container ->20  30  66
Number of data in container  =3
```

Figure Q3.1

[20 Marks]

End of Paper